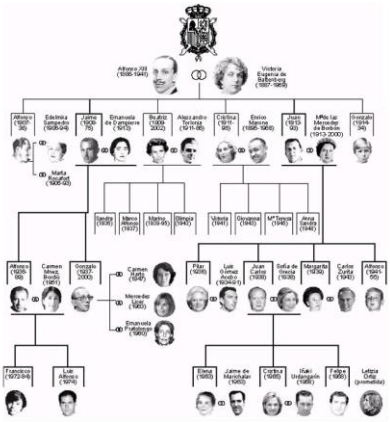
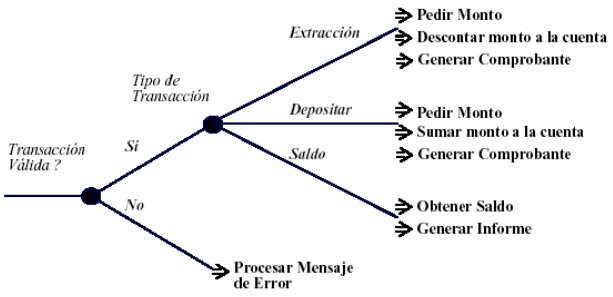
Arboles

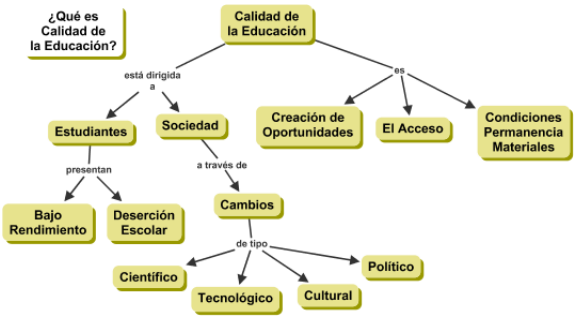
El inconveniente de las estructuras de datos dinámicas lineales es que cada elemento sólo tiene un elemento siguiente, es decir, sólo puedo moverme una posición.
Los árboles representan estructuras no lineales y dinámicas.
Un árbol se define como un conjunto finito de elementos llamados nodos que guardan entre ellos una relación jerárquica.
Existe un nodo diferenciado llamado raíz del árbol, y los demás nodos forman conjuntos diferentes cada uno de los cuales es a su vez un árbol, a estos árboles se los denominó subárboles.
Del nodo raíz van a salir las ramas que van conectando las ramas inferiores.
Un árbol vacío es aquel que no tiene ningún nodo.



Con esta estructura de datos podemos realizar varias aplicaciones, como ser:

- Registrar la historia de eventos
- Árboles genealógicos
- Análisis de circuitos eléctricos
- Fórmulas matemáticas
- Numerar capítulos y secciones de un libro





Conceptos

NODO: Es cada uno de los elementos del árbol
RAÍZ: Es un nodo especial del que descienden todos los demás nodos. Este nodo es el único que no tiene "padre".
HOJA (o nodo terminal): Es donde termina el árbol, no tiene ningún descendiente (nodos
HIJO: Cada nodo que no es hoja y tiene debajo de él 1 o varios nodos

PADRE: Todo nodo excepto la raíz, tiene asociado un ÚNICO nodo predecesor al que llamaremos padre.
HERMANO: Son los nodos que son hijos de un mismo padre .
NODO INTERNO: Cualquier nodo que no sea una hoja.
NIVEL: Cada nodo de un árbol tiene asignado un número de nivel superior en una unidad a su padre.
CAMINO: Una sucesión de enlaces que conectan dos nodos.
RAMA: Camino que termina en una hoja.
PROFUNDIDAD (altura) DE UN ÁRBOL: Es el máximo número de nodos de una rama
PESO DE UN ÁRBOL: Número de nodos de un árbol.
BOSQUE: Colección de 2 o más árboles.

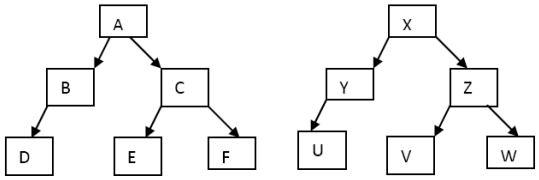
Arboles Binarios

¿Qué es un Árbol binario?

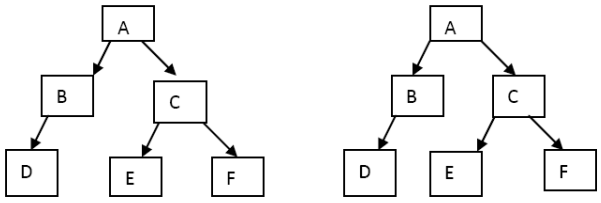
Un árbol binario es una estructura de datos en la cual cada nodo siempre tiene un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno. Usos comunes de los árboles binarios son los árboles binarios de búsqueda, los montículos binarios y Codificación de Huffman.

Tipos de Árboles

Árboles binarios similares



Árboles binarios equivalentes o copias



Árboles binarios equilibrados:
Sus alturas, profundidades o pesos (dependiendo el criterio) se diferencian como máximo en 1 unidad.

Árboles binarios llenos
Un árbol binario lleno es un árbol en el que cada nodo tiene cero o dos hijos, es decir su factor de equilibrio es 0.

Árboles binarios perfecto:
Un árbol binario perfecto es un árbol binario lleno en el que todas las hojas (vértices con cero hijos) están a la misma profundidad (distancia desde la raíz, también llamada altura).

Representación de árboles binarios en memoria

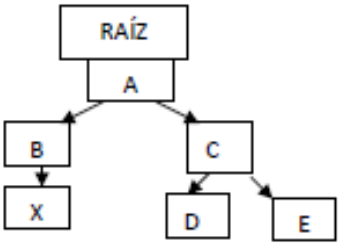
- Representación enlazada o de punteros
- Representación secuencial o con arrays

El principal requerimiento para cualquier representación es que para cualquier árbol T se tenga acceso directo a la raíz de dicho árbol y dado un nodo N del árbol se tenga acceso directo a sus hijos.

Representación enlazada o de punteros

El árbol se representa por medio de una lista enlazada especial en la que cada nodo tiene 3 campos:
- IZQ; campo enlace o puntero al hijo izquierdo del nodo, si dicho hijo no existe, contiene NIL.
- DER; campo enlace o puntero al hijo derecho.
- RAIZ; variable que apunta al nodo raíz del árbol.

Un árbol estará vacío cuando RAIZ <--- NIL.
Para las inserciones y borrados utilizaremos la lista DISP, va a ser una lista enlazada normal en la que cada nodo tendrá los 3 campos siendo el IZQ el campo enlace.



A B C X D E

Representación secuencial

Con 3 arrays: INFO, DER y IZQ y una variable puntero RAIZ.
Cada nodo vendrá representado por una posición K tal que la posición K de INFO, INFO[K], contiene la información de K.
IZQ[K] --> contiene la posición del array donde está almacenado el hijo izquierdo de K.
DER[K] --> contiene la posición del array en la que está almacenado el hijo derecho de K.

Para las inserciones y borrados utilizaremos la lista DISP que enlazará las posiciones del array. La variable puntero DISP apuntará a la primera posiciones libre del array y el resto de las posiciones se enlazarán entre si mediante el campo IZQ.

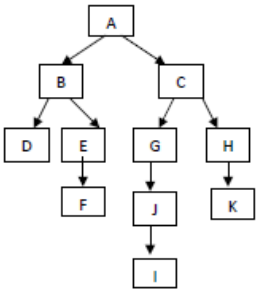
Con 1 array: sea T un árbol binario lleno a casi lleno, para este tipo de árbol lo más eficiente es utilizar un único array para almacenarlo siguiendo las siguientes normas:

El array lo llamaremos **ARBOL** y en **ARBOL**[1] meteremos la raíz del árbol.

Para cada nodo N que ocupe la posición K del árbol, su hijo izquierdo se encuentra almacenado en la posición (2 * K) y su hijo derecho en la posición (2 * k + 1).

Para poder mantener un árbol con esta representación si la profundidad del árbol es h el array en el que se almacena tiene que tener como mínimo 2 h -1 posiciones porque ése es el númeromáximo de nodos que puede tener un árbol lleno de profundidad h.

El inconveniente de esta representación es que si no está lleno se pierde mucho espacio y que además no podemos insertar bajando de nivel porque no hay profundidad.



Profundidad = 5

2 5 - 1 = 31 posiciones

A B C D E G H F J K I

Recorrido de árboles binarios

Recorrido de un árbol es el proceso que consiste en acceder una sola vez a cada nodo del árbol; hay 3 maneras estándar de recorrer un árbol y en las tres se dan los mismos pasos y también coinciden en que se recorre primero su árbol izquierdo y luego el derecho.

Las tres maneras se diferencian en el momento que se accede al nodo raíz.

1. – PREORDEN: 1º se procesa la raíz.
2º subárbol izquierdo.
3º subárbol derecho.

2. – INORDEN: 1º subárbol izquierdo.
2º se procesa la raíz.
3º subárbol derecho.

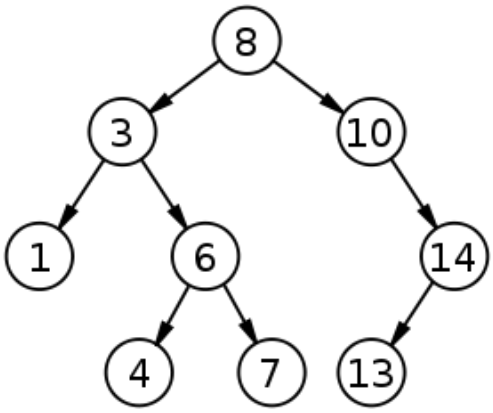
3. – POSTORDEN: 1º subárbol izquierdo.
2º subárbol derecho.
3º se procesa la raíz

Árboles binarios de búsqueda

Un árbol binario de búsqueda también llamados BST (*acrónimo del inglés* ***Binary Search Tree***) es un tipo particular de árbol binario en el que los elementos están organizados de tal manera que facilitan las búsquedas

El árbol depende de un campo **clave** que es distinto para cada nodo y está ordenado respecto a ese campo, tal que si T es un árbol binario, es de búsqueda si cada nodo N del árbol tiene la siguiente propiedad: "*El campo* ***clave de n*** *es mayor que cualquiera de los campos clave de los nodos del subárbol izquierdo de n y de la misma manera el campo clave de n es menor el campo clave de todos los nodos que se encuentran en el subárbol derecho de n"*

Esto supone que si hago un recorrido **inorden** del árbol obtendremos sus elementos ordenados por el campo clave en orden ascendente.



NOTA: Para una fácil comprensión queda resumido en que es un árbol binario que cumple que el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores.

Dos árboles binarios de búsqueda *pueden no ser iguales* aun teniendo la misma información, va a depender del orden en el que ingrese la información.

- *Búsqueda de un elemento*

El algoritmo localiza la posición PTR del nodo que contiene el valor del elemento y en PAD dejamos la localización del nodo padre de PTR, la filosofía del algoritmo consiste en comenzar comparando el elemento con la raíz del árbol y luego ir recorriendo dicho árbol por la derecha o por la izquierda según si el elemento es mayor o menor que el nodo que estamos examinando y así hasta encontrar el elemento o llegar a una hoja del árbol.

- *Inserción de un elemento*

1º Mirar si hay espacio disponible en la lista DISP.

2º Ver si el elemento está en el árbol y si no está, localizar la posición dónde debe ser insertado, para lo que utilizaré el algoritmo de búsqueda tal que PAD será el nodo al que hay que enganchar el nodo que insertamos.

3º Averiguar si el debe insertarse como el hijo derecho o izquierdo de PAD.

4º Finalmente, el nodo tiene que quedar como una hoja.

- *Eliminación de un elemento*

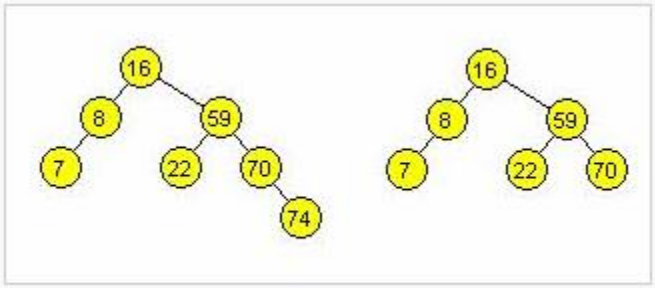
Con el elemento lo primero que hay que hacer es controlar la situación de *UNDERFLOW*, después buscar el nodo y una vez encontrado el nodo se pueden dar 3 casos:

1. - que el nodo no tenga hijos: simplemente se borra y se establece a nulo el apuntador de su padre.

2. - que el nodo tenga un hijo: se borra el nodo y se asigna su subárbol hijo como subárbol de su padre.

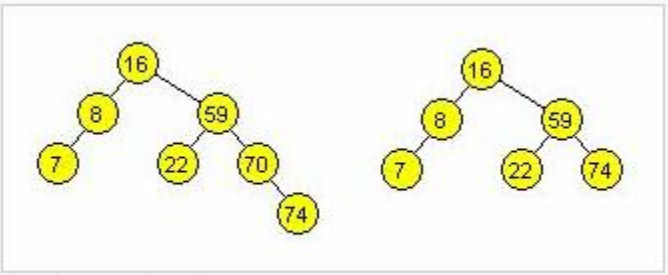
3. - que tenga dos hijos: la solución está en reemplazar el valor del nodo por el de su predecesor o por el de su sucesor en inorden y posteriormente borrar este nodo. Su predecesor en inorden será el nodo más a la derecha de su subárbol izquierdo (mayor nodo del subarbol izquierdo), y su sucesor el nodo más a la izquierda de su subárbol derecho (menor nodo del subarbol derecho).

- *Borrado sin hijos*



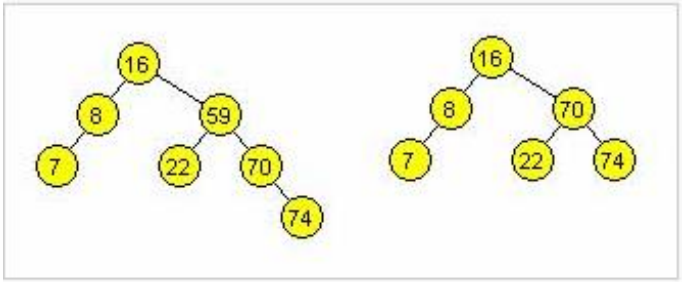
Nodo a eliminar 74

- *Borrado con un hijo*



Nodo a eliminar 70

- *Borrado con 2 hijos*



Nodo a eliminar 59

Aplicaciones de Árboles Binarios

- *Reconocimiento de patrones*

Una de las aplicaciones más importantes de los árboles es utilizarla en el reconocimiento de patrones que es una de las ramas de la inteligencia artificial y en general en la mayoría de las aplicaciones en que se necesita hacer búsquedas.
La filosofía de reconocimiento de patrones es llegar a una conclusión al ir moviéndonos en el árbol según unas determinadas pistas.
Se pueden representar muchos patrones en forma de árboles binarios. El truco está en asignar los significados adecuados a los movimientos de ir hacia la izquierda o la derecha.
El inconveniente es que como lo que usamos son árboles binarios, estas relaciones de búsqueda o patrones sólo pueden ser binarias.

- *Transformación de una notación en otra*

La transformación de una notación a otra se puede realizar estructurando los exponentes como un árbol binario tal que un recorrido sea notación infija, un recorrido preorden la notación prefija y uno postorden la postfija.
La primera tarea es construir el árbol teniendo en cuenta que cada subárbol tendrá como raíz un operador y los hijos o son operados u otros subárboles.
Para equilibrar 0 + a (b + c * d)

Consideraciones de Complejidad

Se definen dos operaciones: Insertar un elemento en el conjunto, y otra que busca y descarta (selecciona) el elemento con valor menor del campo prioridad.

Es necesario encontrar una nueva estructura, ya que las vistas anteriormente tienen limitaciones.
Una lista ordenada en forma ascendente por la prioridad, permite seleccionar el mínimo con costo **O(1)**. Pero la inserción, manteniendo en orden tiene costo promedio **O(n)**; si encuentra el lugar para insertar a la primera es costo 1; pero si debe recorrer toda la lista, debe efectuar n comparaciones; en promedio debe efectuar un recorrido de n/2 pasos.

Una lista no ordenada, tiene costo de inserción **O(1)**, y búsqueda **O(n)**.

En ambos casos la repetición de n operaciones sobre la estructura da origen a complejidad n^2.

Estudiaremos usar la estructura de un árbol binario, ya que ésta garantiza que las operaciones de inserción y descarte sean de complejidad $O(\log_2 n)$, pero deberemos efectuar modificaciones ya que en una cola de prioridad se permiten claves duplicadas.
Las operaciones en la estructura Colas de Prioridad tienen una complejidad Logarítmica.

Una búsqueda binaria sin éxito se comporta como $O(\mathrm{N} \log_{\square} n)$

Una búsqueda secuencial sin éxito el algoritmo se comporta como $O(N)$
Crecimiento de los números armónicos responde a una complejidad Logarítmica